September 1984

MTR9531

J. K. Millen
C. M. Cerniglia

Computer Security Models

CONTRACT SPONSOR OUSDRE/C³I & ESD/ALEE
CONTRACT NO. F19628-84-C-0001
PROJECT NO. 8804C & 4610
DEPT. D-75

MITRE
The MITRE Corporation
Bedford, Massachusetts

Department Approval: Peter S. Tasker
Peter S. Tasker

MITRE Project Approval: Jonathan K. Millen (5504C)
Jonathan K. Millen
Project Leader 8804C

Maureen H. Cheheyl
Maureen H. Cheheyl
Project Leader 4610

ABSTRACT

The purpose of this report is to provide a basis for evaluating security models in the context of secure computer system development. A number of existing models are summarized, and some general considerations for designing and using security models are presented.

TABLE OF CONTENTS

TABLE OF CONTENTS (Concluded)

LIST OF ILLUSTRATIONS

LIST OF TABLES

SECTION 1

EVALUATING MODELS

The purpose of this report is to provide a basis for evaluating security models in the context of secure computer system development for DoD applications. A number of existing models are summarized, and some general considerations for designing and using security models are presented. A new model is also presented, addressing the security policy for A1 systems as defined in the DoD Trusted Computer System Evaluation Criteria.[CSC83].

A paper by Landwehr discusses the role of formal security models and surveys several basic ones [LAN81]. The present report overlaps with Landwehr's somewhat in the choice of models, but we have included a number of more recent models, with an emphasis on concrete ones, and the models are presented here in more detail.

The models surveyed in this report are listed by category in Table I. In the "General Models" category are three models that are used to express security policies in a general way, without architectural assumptions that would force them into any of the other categories. Consequently, these models can be used to prove results or state requirements that apply to all of the more concrete models, and thus help to evaluate them.

REPORT OUTLINE

The remainder of Section 1 discusses the characteristics of security models in general, with brief references to a few models to illustrate the general concepts.

Section 2 has the model summaries. For each model there are a few pages that briefly present its main concepts and requirements. The presentation emphasizes the formal content of the model, such as entities, relations, and axioms, but mathematical symbology has mostly been avoided.

It is anticipated that, in the near future at least, secure system developers will often not be able to find an existing model that fits their needs exactly. A reasonable course of actions is to find one or more models that are close, and modify and combine them as necessary. This sort of activity must be carried out with caution if the desired security features of the chosen models are to be retained and improved on; some considerations for this are discussed in Section 3 of this report.

Multilevel Operating Systems

Bell-LaPadula Model
Biba's Integrity Model
Dion's Protection Model
Spadoc

Database Management Systems

Navy DBMS Security Model
I.P. Sharp Protected DBMS Tool
MAC Multilevel Secure DBMS
SDC Secure Data Management System

Message Handling Systems

NRL Secure Military Message System

Networking

DoDIIS/DNSIX

Guards

ACCAT Guard
LSI Guard
Message Flow Modulator

General Models

SRI Model of Multilevel Security
Rushby's Separability Model
Goguen-Meseguer Non-interference Model

Table I. MODEL CATEGORIES

Section 4 presents a new model, addressing the requirements for DoD trusted system evaluation [CSC83]. It resembles the Bell-LaPadula model, but omits Multics-specific features such as the directory hierarchy and functional rules. It is more restrictive in that it incorporates discretionary security into the mandatory security level and has a more detailed treatment of trustedness that involves integrity and limited privileges.

Conclusions and recommendations are given in Section 5, in the form of questions and answers about security modeling.

A glossary of terms used in the context of computer security is included as an appendix.

THE USE OF MODELS

A security model is only a part of the process of secure system development. One paradigm is illustrated in Figure 1, showing four principal manifestations of system security: as a policy expressed informally (at least from a mathematical point of view), a formal security model, a formal top level specification, and an implementation of the system. This view focusses on the formal aspects of the development, of course; one can also expand the implementation area considerably, dividing it into several stages or components, such as PDL (program design language), high-order language software, object code, microcode, and hardware, etc.

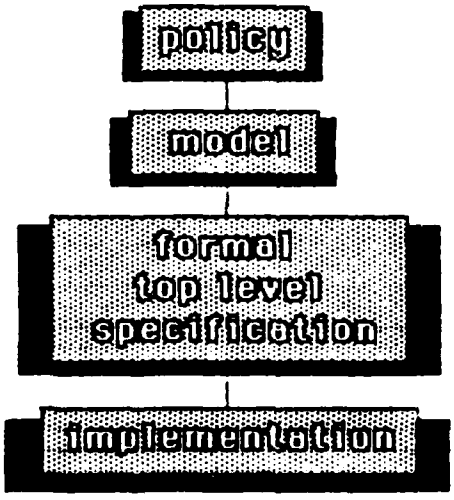


Figure 1. STAGES OF SECURE SYSTEM DEVELOPMENT

In looking at Figure 1, one should observe not only the boxes but also the lines joining them. Successive representations of security should correspond, and those correspondences should be justified as formally and completely as possible. An important question about a formal model, in particular, is how it will be used to evaluate the formal specification. The answer depends as much on the formal specification as on the model, so we will not be able to pursue it in this report.

MODEL CHARACTERISTICS

Computer security models are engineering models, giving them somewhat more freedom than models used in physical science. In physical science, reality comes first, and one uses a model to make predictions about physical events and measurements. If a prediction fails, the model is wrong. In engineering, the model comes first. The engineer decides what the system ought to do, and then constructs a system that does it. If the system output does not match the model, the system is wrong, not the model.

This is not to say that engineering models cannot have mistakes. They can be internally inconsistent, or they may fail to meet user needs. Or they may be unimplementable. After all, models portray a kind of perfection that is not achievable in the real world. A model is not useful unless one can construct a real system whose relevant behavior matches that of the model, to within some needed precision and reliability.

User needs are expressed in the form of security requirements. Security requirements are imposed at various levels, from various sources, and it is the job of a model to bring them together in a way that is meaningful for design guidance. There are overall requirements for handling classified information, and more specific policies applicable to computer systems, such as DoD Directive 5200.28 and the corresponding manual. These sorts of requirements lead to the general form of a model as having subjects with clearances and objects with levels, as well as the classification-compartment structure of levels, and the notion of access permission.

The application and environment of a system plays an important part in setting additional security requirements. The need for auditing, sanitization, downgrading, and special privileges for system security officers necessitate more complex models. Other sources of special requirements are the interface of the computer

system with users, via printed output and display terminals, and with other systems over network connections.

SELECTIVITY

Models are generally selective, in the sense that there are some aspects of security in the real system that are not mentioned in the models. This is inherent in the nature of a model: it represents certain features of interest, leaving out others, for the sake of clarity.

One type of selection has to do with coverage of the trusted computing base. The phrase "Trusted Computing Base", abbreviated "TCB", applies to the totality of components of a system that are trusted, or security-critical: object manager, policy manager, and any trusted processes. It may also refer to hardware components essential to security. As the discussion of interfaces will bring out, a model may cover some of the security-critical components of a system, but not all. Many models, for example, do not cover the specific requirements for trusted processes in a given system. In this case, one should look for other models to cover the missing portions.

The more abstract models are selective because they focus on single types of security requirements. Simplicity is necessary in a model whose principal purpose is to derive the mathematical consequences of general protection mechanisms. Models of information flow and of the propagation of access rights are limited in this way.

WEAKNESSES IN MODELS

Models can have weaknesses in them, just as formal specifications can, leading to security flaws in the final system. Three examples will be mentioned here to illustrate the pitfalls in designing, or redesigning, models. The first two models involved are summarized in later sections of this report.

There is a rule in the Bell-LaPadula model called "change-subject-current-security-level" by which a subject can change its current security level to any value that will satisfy the simple security property and *-property. In particular, it is possible for a subject to downgrade itself, as long as its read accesses are only to objects at or below the new, lower level.

The channel works as follows. When the subject is at the initial, higher level, it can read one bit of classified information at that level, and then decide to get read access to a fixed lower-level object or not, depending on the value of that bit. After releasing read access to the higher-level object, it can then downgrade itself to the lower level. At this point, the subject is not supposed to know any higher-level information; yet, it can determine the value of the higher-level bit by testing whether it does or does not have read access to the lower-level object. The result can then be written into some other lower-level object.

The problem is more serious than its single-bit version suggests. First, it might be possible to repeat it rapidly, perhaps hundreds of times per second, with successive bits. One can also devise more complex versions involving several lower-level objects, to transmit larger words at a time.

This channel works despite the assumption that the subject is "memoryless", i.e., it has no implicit memory of its own. The information has not been stored in the subject, but rather in the state of the system.

Of course, one could prevent this channel by requiring that the subject forget its accesses when it is downgraded. This is quite correct, but it is unfortunately excluded by the rule, which states that the access set "b" is unchanged.

Technically, a system can still avoid the channel and obey the model if it never permits a change-subject-current-security-level call by itself, but only allows the call in a compound request in which the subject first releases its accesses. But this device defeats what is probably the best argument in favor of having specific rules, namely, that they might act as design guidance to avoid security problems.

The problem that will serve as a second example almost happened, but was discovered in an early draft of the NRL SMMS model. The CCR (Container Classification Required) attribute of containers is supposed to prevent the otherwise allowable access of a lower-level user to an object of equal level inside a higher-level container, when the object is referenced indirectly - by its relative position inside the container. However, there was originally no axiom that disallowed a service that translated indirect references to direct ones, enabling users to bypass the CCR restriction. When this was noted, an new axiom was added: "Translating Indirect References."

A third example is from the draft Integrated Computer Network (ICN) model [FEI81]. In the ICN model, there is a novel extension of the notion of a security level. In this model, one security level is actually a set of basic levels, in order to reflect indeterminacy or lack of knowledge of the level, pending computations that will determine the correct assignment. Presumably at least one of the basic levels in the level set is correct.

The "upward flow policy" in the ICN model requires that when data is transferred from one object to another, the destination level set dominates the source level set according to a definition that would be satisfied, in particular, if there were a correspondence between the two level sets such that the destination levels each dominated their corresponding source levels.

After a data transfer operation, one would expect that the destination level set contains a level that dominates the correct levels of both the source and destination objects, since the latter now has both levels of data. However, this is not the case in this example:

Source Level Set	Destination Level Set
(C,{NSI},SECURE)	(S,{NSI},SECURE)
(C,{RD},SECURE)	(S,{RD},SECURE)

(C = Confidential, S = Secret, NSI = National Security Information, and RD = Restricted Data; SECURE is a "partition name".)

Specifically, the problem is that the source data might have been (C,{NSI},SECURE) while the destination data might have been (S,{RD},SECURE). The data transfer should not be allowed if the destination level set does not have a level dominating (C,{RD,NSI},SECURE). The upward flow policy allows this transfer, however, so it evidently needs to be changed, for general application.

The actual ICN system does not permit arbitrary level sets; it implements only one type of level set, called PARD. A PARD level is a set of basic levels with the same category set, all in the SECURE partition. The problem above is eliminated by this restriction, since the level sets used have multiple category sets and are consequently not PARD levels.

INTERFACES

One of the reasons that the more concrete models differ from one another is that they are modelling different interfaces of, or within, a secure computer system. It should not be surprising that a model of a relational DMS, with powerful operations for forming views, looks different from a model of a general-purpose operating system kernel, which controls access on a single-word basis to virtual memory segments.

When looking for a model, therefore, the first question should be, "a model of what?" Our first task is to provide some concepts for phrasing the possible answers. We begin with what is hoped to be a general picture of a secure computer system, shown in Figure 2. This is a single-computer system. If we are considering a network or distributed system, this picture represents one host or processor in it.

The diagram proceeds from the notion of "per-process virtual environments," a widely accepted conceptual scheme for security. There are processes that provide services such as data management, electronic mail, text editing, and programming in various languages. Programs, data files, messages, and other objects are accessed by these processes through a shared facility labelled "object manager" in the figure.

We will not be concerned here with the boundary between hardware and software; the figure provides only a functional view of the system. Any attempt to draw a line between hardware and software functions would be frustrated by the variety of tradeoffs between the two in different systems, and confused by the role of firmware or microcode. It is worth noting, however, that security is practical only in systems having built-in features to support memory protection, such as segmentation registers, or the control of access to data objects, such as capabilities or object descriptors.

An object manager will permit or deny access to an object by process according to its current "access state". All requests from processes to change the access state are handled by the component labelled "policy manager." The policy manager has an interface with each process to accept state change requests and deliver replies, and an interface with the object manager to control its state.

The policy manager and the object manager together are often referred to by the phrase "security kernel". A kernel generally includes some basic operating system services; those operating system services that are not security-critical are assumed to be performed within processes.

The security architecture of a number of systems also includes "trusted processes" for certain security-critical tasks. For now, we simply note that such processes may exist in a given system.

Let us return now to the notion that a model describes an interface. We can use Figure 2 to draw the interfaces described by different models, and thus categorize them. We begin by drawing the interface of the TCB, in Figure 3.

Access control models describe the policy manager interface, as indicated in Figure 4. The Bell-LaPadula model is in this category. There are, of course, important distinctions to make within this class, such as level of detail and the nature of the policy.

The principal weakness of access control models is evident from this picture. Their interface is inside, and different from, the interface of the TCB. In other words, they describe only a part of the security-critical behavior of the system.

Application-oriented models typically describe the interface to a program that supports a service, such as a DBMS or message system. This program is run within the user process; consequently the interface divides the user process, as suggested in Figure 5. Note that the user process has the option of addressing the kernel directly, although it may not be expected to do so.

Application-oriented models may also be at higher or lower levels. The I.P. Sharp DMS model is actually a kernel interface model, in which the policy and object manager interfaces have been redesigned to handle primitive relational operations. By contrast, the NRL SMMS model seems to describe the system interface with people at terminals. We can add terminals to the picture as in Figure 6 - each terminal is connected to the process that performs I/O for it. Terminals are really objects that must be accessed through the object manager, but they are (normally) not shared by different processes.

With the topic of message systems we must consider the connection of host computers into networks. A port to a network node or network interface processor is a protected object that is typically accessible only to trusted software. A model may or may not deal with the interface between the host computer and the network. As a rule of thumb, we might expect that a model of a network of homogeneous hosts would treat network interfaces as an invisible implementation detail. This gives a picture like Figure 7. Recent discussions on the requirements for SACDIN have taken the former approach, since all SACDIN nodes share a common design.

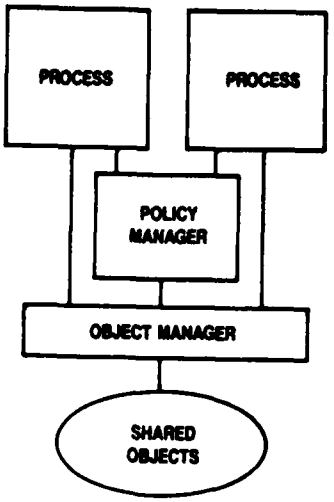


Figure 2. TYPICAL SECURE SYSTEM

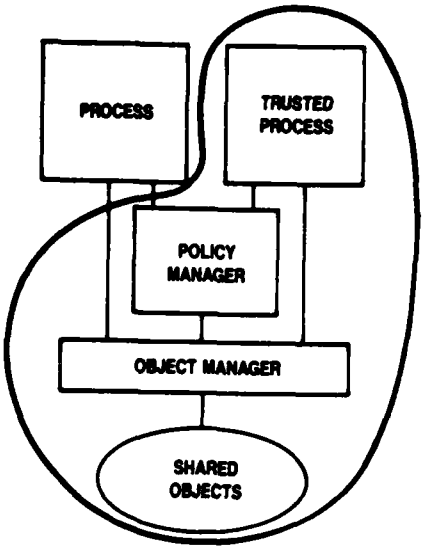
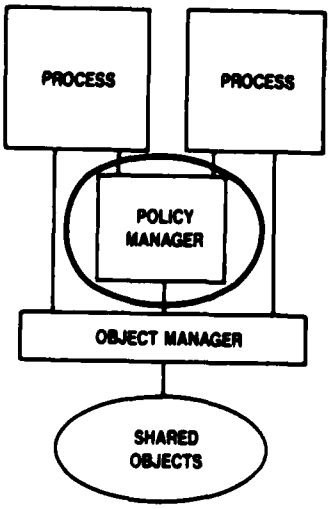


Figure 3. TYPICAL TCB INTERFACE



IA-69,021

Figure 4. POLICY MANAGER INTERFACE

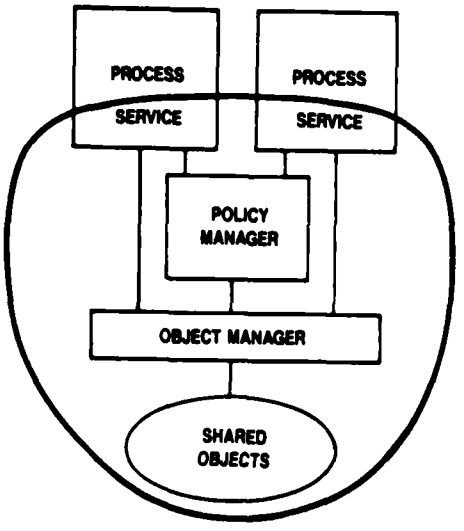


Figure 5. APPLICATION INTERFACE

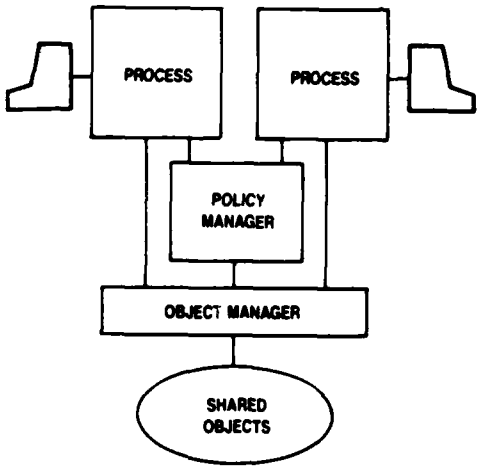


Figure 6. PROCESS-LOCAL TERMINALS

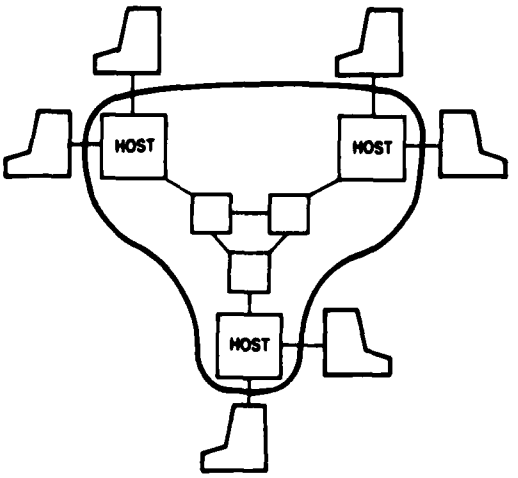
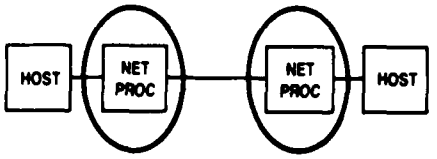


Figure 7. HOMOGENEOUS NETWORK



IA-89.022

Figure 8. NETWORK PROCESSOR INTERFACE

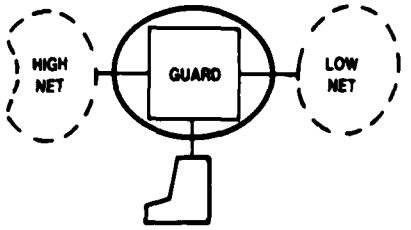


Figure 9. GUARD INTERFACE

On the other hand, a model for a secure host computer that must connect with other systems not assumed to satisfy the same model should specify the network interface. This means that any security-critical assumptions about network packets, such as the existence of a trustworthy security label, are specified.

The DNSIX model for DoDIIS is an interesting example of a case where the model interface is to several hosts in a network. The intent of the DoDIIS security architecture is to set up a standardized interface between various types of hosts and the DDN (Defense Data Network). Functionally, DNSIX stands between a local host and other hosts assumed to be connected with it through DDN, as illustrated in Figure 8. The DNSIX model is for a policy manager that deals with connection establishment requests from either the local or any remote host; it also is concerned with audit trail recording of network activity at its local host.

A guard is a system that stands between two others, monitoring information flowing from one system to the other, according to some given release policy. The policy may be algorithmic, in which case the guard stops information that does not satisfy the given test. Or the policy may require manual release by a security officer who must view the information and make a decision. In the latter case, the guard has an interface with the officer's terminal, as shown in Figure 9.

SECTION 2

MODEL DESCRIPTIONS

Each presentation in this section summarizes a model with regard to any existing information in the following areas:

Sources: Documents referenced as source material for the model description.

Overview: An introductory discussion sometimes necessary to point out unique features or objectives of the model.

Entities: Defining the entities in the model, such as subjects and objects; their attributes, such as security levels; and changing system status information such as current accesses.

Operations: In some models, the system state can be transformed, or data transferred, only by means of a specific set of operations, which will be listed.

Axioms: Security policy, criteria, properties, constraints, restrictions, or other limitations on the operation of the system.

Remarks: Any remaining comments.

BELL-LAPADULA MODEL

Sources

D. Elliott Bell and Leonard J. LaPadula, "Secure Computer Systems: Mathematical Foundations," ESD-TR-73-278 Vol. I, AD 770 768, The MITRE Corporation, Bedford, Massachusetts, 1 March 1973.

Leonard J. LaPadula and D. Elliott Bell, "Secure Computer Systems: A Mathematical Model," ESD-TR-73-278 Vol. II, AD 771 543, The MITRE Corporation, Bedford, Massachusetts, 31 May 1973.

D. Elliott Bell, "Secure Computer Systems: A Refinement of the Mathematical Model," ESD-TR-73-278 Vol. III, AD 780 528, The MITRE Corporation, Bedford, Massachusetts, December 1973.

D. Elliott Bell and Leonard J. LaPadula, "Secure Computer System: Unified Exposition and Multics Interpretation," ESD-TR-75-306 (MTR-2997), The MITRE Corporation, Bedford, Massachusetts, July 1975.

Overview

The model evolved through three significantly different forms in Volumes I - III. The Unified version is very close to that in Volume III, with a final selection and renumbering of rules. Volume I had only one (simple) "security property" and no transition rules. Volume II introduced the *-property and ten rules. Volume III introduced the directory hierarchy and used it to dispense with "control" access. The distinction between an object's current and maximum security levels also appears in Volume III, leading to a different form of the *-property.

Entities

- SUBJECTS: Active entities, i.e., users or processes. Users and trusted processes are considered to be trusted subjects. Subjects are assumed to be memoryless.

- OBJECTS: Passive entities which are protected repositories of information.

- SECURITY LEVELS:

  A security level consists of a classification and a set of categories.

Classifications: Top Secret (TS), Secret (S), Confidential (C), Unclassified (U). These are ordered, TS being the greatest.

Category: A formal need-to-know designation, for example, NUCLEAR, NATO, CRYPTO, etc.

Security levels are partially ordered. One level is greater than or equal to another if the classification of the first is greater than or equal to that of the second, and the category set of the first includes that of the second.

- ACCESS STATE

The system state is expressed as a set of four components, (b,M,f,H).

The first component, b, is the current access set which is a set of triples of the form:

(subject, object, access-mode)

This means that subject has current acess to the object in the specified access mode.

A subject can have access to an object in the following modes:

e	execute	(for running programs)
r	read	(observation with no alteration)
a	append	(alteration with no observation)
w	write	(both observation and alteration)

The M component is an access permission matrix. The matrix records the modes in which a subject is permitted to access an object. Thus, the entries of the matrix are subsets of the set of access modes.

The f component of a system state is a level function, which gives the security level assignments for subjects and objects. A subject has both a maximum level and a current level; an object has just one level.

The H component of a system state is the hierarchy; a hierarchy is a directed rooted tree of objects. Objects not in the tree are "inactive" and not accessible. The model operations preserve a property of the hierarchy called compatibility: the security level of each object dominates that of its parent.

Operations

There are eleven "rules", R1-R11, corresponding to typical kernel requests as found in the Multics system. The rules alter the system state and are required to preserve the security properties. Inputs to the system are called requests and outputs from the system are called decisions. The rules may be divided into the following function groups:

Altering current access: (Access = read, append, execute, or write)

GET Access: (R1-R4). To initiate access to an object by a subject in the requested mode.

RELEASE Access: (R5). The inverse of GET Access; terminates access.

Altering access permission: (Access = read, execute, write, or append)

GIVE Access: (R6). To allow a subject to extend the designated access permission to another subject.

RESCIND Access: (R7). The first subject must have write access to the parent of the object in the hierarchy. The inverse of GIVE Access. Rescinding permission has the side effect of forcing access release.

Altering the hierarchy:

CREATE OBJECT: (R8). To activate an inactive object; i.e., attach an object to the current hierarchy. A newly activated object is considered to be purged of data. This rule preserves compatibility.

DELETE OBJECT: (R9). To deactivate an active object; i.e., to detach from the hierarchy an object and all other objects beneath it in the hierarchy. Deleting an object has the side effect of releasing all accesses to it.

Altering the level function:

CHANGE SUBJECT CURRENT SECURITY LEVEL: (R10). To allow a subject to alter its current security level, within its maximum level.

CHANGE OBJECT SECURITY LEVEL: (R11). To allow a subject to upgrade the security level of some object, within the subject's maximum level.

Axioms

The rules have been shown to constrain the operations to obey the following properties:

Simple Security
Property: A subject may have read or execute access to an object only if the security level of the object is less than or equal to the maximum level of the subject.

*-Property: An untrusted subject may have append access to an object only if the security level of the object is greater than or equal to the current security level of the subject;

An untrusted subject may have write access to an object only if the security level of the object is equal to the current security of the subject; and

An untrusted subject may have read access to an object only if the security level of the object is less than or equal to the current security level of the subject.

The purpose of the *-property is to prevent untrusted software from compromising classified information by copying it into an object of lower security level.

Discretionary Security
Property: Every current access is present in the access permission matrix.

Volume II also mentions a Tranquility Principle, that the classification of active objects will not be changed during normal operation.

BIBA'S INTEGRITY MODEL

Source

K. J. Biba, "Integrity Considerations for Secure Computer Systems", The MITRE Corporation, Bedford, MA, MTR-3153, 30 June 1975.

Overview

Unlike other models, this model does not address unauthorized disclosure of information; it is concerned only with preventing unauthorized modification.

Entities

- SUBJECT: any system element which performs information accesses.

- OBJECT: any system element accessed.

- INTEGRITY LEVELS:

  C-Crucial; VI-Very Important; and I-Important.
  These integrity classes are directly analogous to the security classes: TOP SECRET, SECRET, and CONFIDENTIAL.

  Integrity levels are partially ordered under leq (which stands for less than or equal). I leq VI leq C.

  Assignment of an integrity level to a subject is limited by the permitted integrity level range of the associated user.

- ACCESS CONTROL LIST:

  An access control list (ACL) is associated with each object.

  A subject can have access to an object in any combination of the following access modes: o (observation), m (modification), and e (execute). In addition, a subject can have I (invocation) access to another subject, representing the ability to perform interprocess communication.

Axioms

The Axioms depend on which of several policies are used.

Non-discretionary Integrity Policy

There are four types of non-discretionary integrity policy:

1. The Low-Water Mark Policy for Subjects: a policy that is dynamic in the sense that the integrity level of a subject is decreased as necessary to the lattice meet (extension of "minimum" to a partial ordering) of the integrity levels of the objects previously observed.

   For each observe (read) access of an object by a subject, the integrity level of the subject after the access equals the minimum of the object's and subject's integrity levels preceding the access.

   A subject is constrained to receive modify access only to those objects which possess an integrity level less than or equal to that of the subject. A subject can gain "invoke" access only to subjects of less or equal integrity.

   Use of this policy makes generalized, domain independent programming awkward since the set of objects modifiable by a given subject can change with each observation as the result of having accessed lower integrity objects. This makes it possible for a subject to sabotage its own processing because objects which are necessary for its own function can become inaccessible.

2. A Low-Water Mark Policy for Objects: a policy which allows the integrity level of modified objects to change. Instead of preventing a subject from modifying higher integrity objects, this policy lowers the integrity of modified objects, if necessary, to that of the subject. This policy allows improper modifications but insures that the improper modifications are apparent.

   A Low-Water Integrity Audit Policy is an unenforced variant of the above model in which integrity levels are fixed. Violations are recorded in an audit trail, from which the "current corruption level", the real integrity level, can be calculated for each object.

3. The Ring Policy: Integrity levels of both subjects and objects are fixed during their lifetimes and only modifications of objects of less or equal integrity are allowed. A subject may invoke another subject only if its integrity level is less than or equal to the integrity level of the one invoked. A subject is permitted to observe objects at any integrity level. A subject needs to be cautious when using data from a lower integrity object.

4. The Strict Integrity Policy: A complement of the Bell-LaPadula security policy consisting of two axioms which, analogously to the simple security condition and *-property prevent the direct and indirect sabotage of information.

   Three axioms characterize this policy:

   Simple Integrity Condition

   > For any subject with observe access to an object the integrity level of the subject must be less than or equal to the integrity level of the object.

   Integrity *-Property

   > For any subject with modify access to an object the integrity level of the object must be less than or equal to the integrity level of the subject.

   Invocation Property

   > If a subject invokes another subject then the integrity level of the second subject must be less than or equal to the integrity level of the first subject.

   Using these axioms, no information may be transferred from objects of low integrity to ones of higher integrity, assuming that information transfer results only from observe and modify accesses.

Discretionary Integrity Policy

There are three discretionary access control policies:

1. Access Control Lists: As defined in the specific characteristics list. The privilege to modify an ACL is defined by the privilege to modify the object in which the ACL is located.

2. Object Hierarchy: a rooted tree of objects: a subset of the entire object set. The ancestors of an object are those on a path between it and the root. To access any object a subject must have observe access to all its ancestors.

3. Rings: A ring is a privilege attribute of a subject. Rings are numbered, with lower-numbered rings representing higher privilege. The access axioms for rings are that:

a. A subject may invoke subjects of greater privilege only through an allowed range of rings; and, of less than or equal privilege indiscriminately.

b. Subjects may only observe objects in an allowed range of rings.

c. Subjects may only modify objects in an allowed range of rings.

DION'S PROTECTION MODEL

Source

L.C. Dion, "A Complete Protection Model," Proc. of the 1981 Symposium on Security and Privacy, IEEE Cat. No. 81CH1629-5, pp. 49-55.

Overview

This model is an advance on the Bell-LaPadula model in that it combines security and integrity attributes, and provides for parametric limitations on trustedness and overclassification of data. It does not specify rules for operations.

Entities

- SUBJECTS: Processes, representing programs executing on behalf of users.

- OBJECTS: Any data storage entities.

- SECURITY ATTRIBUTES:

  Each subject and object has three security levels and three integrity levels associated with it. Security levels are partially ordered, and so are the integrity levels.

  A subject's absolute security and integrity levels are the levels at which the associated user is operating.

  A subject's read security level is the highest from which it can read; the read integrity level is the lowest from which it can read. These levels will differ from the absolute levels only if the subject is trusted.

  A subject's write security level is the least to which it can write; the write integrity level is the greatest to which it can write. These levels will differ from the absolute levels only if the subject is trusted.

  An object's absolute security and integrity levels represent the sensitivity of the data in the object.

  An object's migration security level is the greatest to which information in the object may flow; the migration integrity level is the least to which information in the object may flow.

An object's corruption security level is the least from which information can flow into the object; the corruption integrity level is the greatest from which information can flow into the object.

- CONNECTIONS:

Information flow is assumed to occur only when there is a (unidirectional) connection from one object to another. Connections result from access requests by subjects.

Axioms

The following consistency properties must be satisfied when a process establishes a connection from an object O1 to an object O2. In this case the process has read access to O1 and write access to O2. These properties are more or less directly implied by the definitions of the various levels given above.

Migration property: The migration levels of O2 must be at least as restrictive as those of O1.

Corruption property: The corruption levels of O1 must be at least as restrictive as those of O2.

Security properties: The subject's read security level must dominate the absolute security level of O1, and its write security level must be dominated by the absolute security level of O2.

Integrity properties: The subject's read integrity level must be dominated by the absolute integrity level of O1, and its write integrity level must dominate the absolute integrity level of O2.

Write/Corruption properties: The subject's absolute security level must dominate the corruption security level of O2, and its absolute integrity level must be dominated by the corruption integrity level of O2.

Read/Migration properties: The subject's absolute security level must be dominated by the migration security level of O1, and its absolute integrity level must dominate the migration integrity level of O2.

SPADOC (SPACE DEFENSE OPERATIONS CENTER)

Source

L. Buczkowski and J. Freeman, "SPADOC 4 Technical Report Part I - Formal Security Document", Final Report CSO-TR394, Ford Aerospace and Communications Corporation, Colorado Springs, Colorado, 3 November 1983.

Overview

This model is reminiscent of Bell-LaPadula in that it has axioms and rules, but it provides for aggregate objects like containers in the NRL model. There is only one mode of access, represented implicitly by the membership of an object in a "domain" associated with a subject.

Entities

- SUBJECTS: Certain basic objects that represent active entities such as users or programs.

- OBJECTS: Entities, which are either basic objects, aggregate objects or domains.

  Basic Object: Data or a subject.

  Aggregate Object: A collection of basic objects. The aggregate object could represent a file record, display, buffer, or message.

  Domain: A collection of aggregate objects. The domain can represent an address space, a user, a separate interfacing system, or a file in a database. There are two types of domains: I elements, domains associated with interfaces; and S elements, domains within the main model.

- SECURITY LABEL:

  Comprised of two components: a classification and a set of categories, possibly empty.

- DISCRETIONARY LABEL:

  A discretionary label is a subset, possibly empty, of elements called discretionary items. A discretionary item may be associated with a group of people or single

individual. Two elements are identified as discretionary items: DL_CHANGER and SL_CHANGER. These elements permit the specification of permissions to change the discretionary label (DL) and security label (SL) mappings.

- MAPPINGS:

A "state" of the model is defined as an instance of four mappings: SL, DL, Owns and Contains.

Security Label Mapping (SL): This mapping is defined on the entire set of entities and maps into the security label set. The security label of a basic object is static.

Discretionary Label Mapping (DL): This mapping is defined on the entire set of entities and maps into the discretionary label set. The mapping DL is static on basic objects.

The following are two Boolean mappings which define the basic relationships between the three types of entities.

Owns: This function is defined on the set of ordered pairs (domain a, aggregate object p) of elements from the Domain set and Aggregate Object set. If Owns(a,p) is TRUE then "Domain a owns Aggregate Object p." Owns is static and each aggregate object is owned by exactly one domain, termed the <u>owner</u> of the aggregate object.

Contains: This function is defined on all ordered pairs (aggregate object p, basic object b) of elements belonging to the Aggregate Object set and Basic Object set. If Contains(p,b) is TRUE, then "aggregate object p contains basic object b". Contains defines the relationship between an aggregate object and a basic object. Each basic object is contained by at least one aggregate object.

Operations

There are seven types of transitions in this model. Each transition changes a mapping, and is initiated by a subject, termed the "initiator".

In transitions involving a domain, the initiator and the given domain must satisfy the "initiator-domain relationship": there is some aggregate object that contains the initiator and is also owned by the given domain.

Transitions:

UNIT_FLOW: This transition passes a basic object from one aggregate object (source) to another (receiver) when both are owned by the same domain, by changing the Contains mapping.

AGGREGATE_FLOW: This transition passes all basic objects from one aggregate object to another aggregate object not necessarily owned by the same domain. In this transition, the Contains, SL, and DL mapping functions are changed.

PURGE: This transition deletes a basic object from an aggregate object, changing the Contains mapping function.

The next four transitions specify conditions under which the security label and the discretionary label of an entity may be changed. The following four transitions are used to specify the reclassification mechanism for containers of information within the system to reflect accurately the security classification and discretionary labeling of the information in them. These transitions are not meant to model downgrading or upgrading of the actual classification of the information itself.

CHANGE_SL/AO: This transition specifies conditions under which the security label of an aggregate object may be changed either up or down.

CHANGE_DL/AO: This transition specifies conditions under which the discretionary label of a given aggregate object may be changed. The preconditions and postcondition are the same as transition CHANGE-SL/AO with ´discretionary label´ replacing each occurrence of ´security label´. The initiator subject must be contained in the same domain as the affected aggregate object.

CHANGE_SL/D: This transition effects a change in the security label of a domain. In changing a label associated with a domain, the protective constraint requires that specific authorization via the discretionary label SL-CHANGER be associated with the transition initiator.

CHANGE_DL/D: This transition effects a change in the discretionary label of a domain. In changing a label associated with a domain, the protective constraint requires that specific authorization via the discretionary label, DL-CHANGER, be associated with the transition initiator.

Axioms

A secure state is defined as one satisfying the following two conditions: ("Dominated by" is the Bell-LaPadula partial ordering.)

a. For all aggregate objects ´p´ and domains ´a´, if ´a´ OWNS ´p´ then SL(p) must be dominated by SL(a) and DL(p) must be dominated by DL(a).

b. For all basic objects ´b´ and aggregate objects ´p´, if ´p´ CONTAINS ´b´ then SL(b) must be dominated by SL(p) and DL(b) must be dominated by DL(p).

NAVAL DBMS SECURITY MODEL

Source

Richard D. Graubart and John P. L. Woodward, "A Preliminary Naval Surveillance DBMS Security Model", The MITRE Corporation, Bedford, Massachusetts, MTR-8475, May 1982.

Overview

This model, like the I.P. Sharp and SDC models, is for a relational data base system. It does not have an analogue of the *-property, but each user can be limited to specific operations.

Entities

- SUBJECTS: users

  Four classes of users are identified in the NSS application:

  Analyst: Able to view, modify, delete, transfer and correlate information in the system.

  Remote-User: Limited to querying the data base via the read (data) operator.

  System Security Officer (SSO): has authorization to change security levels of data and clearances of users.

  Data Base Administrator (DBA): has authorization to define data bases.

- OBJECTS: The objects in this model include:

  Device: A piece of physical hardware, used for input or output purposes and excluding those mediums used for primary or secondary storage of data.

  Data Base: A collection of relations.

  Relation: A two-dimensional array of elements. Can be viewed either as a set of fields or a set of records.

  Field: A sequence of data elements; one column of a relation.

  Record: A sequence of data elements; one row of a relation.

Data Element: An intersection of a field and record. An atomic element of the data base.

All of the above, except for devices, are referred to as data objects. Data bases, relations, fields and records are special forms of objects known as containers.

- SUBJECT ATTRIBUTES:

Subjects are assigned two security levels: The first is a clearance which is considered a maximum security level. Secondly, each user has an access level, a current security level which varies from system low to the user's clearance.

Operator Authorization List: Associated with each user, this list indicates which operators a user can use.

- OBJECT ATTRIBUTES:

Security Levels: There two kinds of security levels: DSL, Default Security Level, and ISL, Implicit Security Level. A DSL may be set explicitly by a user; otherwise the DSL of its container applies. An explicit DSL is mandatory for data bases and devices. An explicit DSL of an object may be less than, greater than or equal to that of its container.

It is possible for the DSL of a field and record to conflict. To prevent this, a user may specify whether the default security level of the field or the record has priority within the relation.

In addition to the DSL, there is also an Implicit Security Level: ISL. An ISL is used to indicate the security level of the most sensitive data in the container or data element. The ISL of a container is equal to the highest DSL within the container and changes as this DSL changes. The ISL of a data element is equal to the DSL of a data element.

Thus, "security level" for this model has two meanings: when doing comparisons of access levels to security levels, security level shall refer to the ISL. When referring to modifying or setting the security level of an object, security level shall refer to the DSL.

ACL: Each object has two types of access control lists associated with it. The ACL for an object specifies which

users have what type of access to the object. The types of accesses valid in a particular ACL depend on the type of object. There is an access type for each operation (see below). The exclusionary ACL (XACL) for an object specifies which users are denied each specific type of access appropriate for the object. If an object does not have an ACL or XACL then the ACL or XACL of its container applies.

Inherent Access: Each data base has an inherent access list, specifying which users have change-access or change-level/read access to it, overriding the ACLs of the objects it contains.

Operations

Operator Class	Operator Name	Applicability
Data Access	Read	All Objects
	Write	All Objects
	Delete	Containers Only
Data Definition	Define-db	Databases Only
	Define-rel	Relations Only
Attribute Changing	Change-level/Read	All Objects
	Change-access	All Objects

Axioms

In order for a user to apply an operator to an object, three requirements must be met:

1. Operator Authorization: The operator must be listed on the user's operator authorization list.

2. Non-discretionary: The access level of the user must dominate that of the object. A user may not upgrade an object above his own level.

3. Discretionary: the object's ACL and XACL and the ACLs and XACLs of the containers of the object in question must grant the user the requested discretionary access to the object or the user has inherent access to the object.

I.P. SHARP PROTECTED DBMS TOOL

Source

M.J. Grohn, "A Model of a Protected Data Management System," ESD-TR-76-289, I.P. Sharp Associates Limited, Ottawa, Canada, June, 1976.

Overview

This model is also for a relational data base system. It assigns a security level (which includes an integrity level component) to a relation as a whole.

Entities

- SUBJECTS: Processes. A process has both a maximum protection level and a current protection level.

- OBJECTS: Objects consist of a permission matrix, a description, and a value.

  A permission matrix is a set of elements, each indicating permission for a subject to either observe or modify the object.

  The description component is motivated by the intended application to relational data bases. An object can be a relation, and its description includes the names and formats of its fields.

  The value component is to be interpreted as the set of tuples in a relation; it could also be the sequence of statements in a program.

  The components of an object may be accessed separately, but all three have the same protection level, that of the object as a whole.

- PROTECTION LEVELS: A protection level has both security and integrity components, each of which has a classification and a category set. The partial ordering of protection levels is based on the partial ordering of security components as in Bell-LaPadula and the (inverted) partial ordering of integrity components as in the Biba strict integrity model.

- DIRECTORY SYSTEM: For each protection level, there is a directory consisting of a set of entries identifying all objects known at that level. An object is represented by at most two entries: one at the lowest level at which it is known, and one at its own level. An entry consists of an object identifier and a code indicating the level of the other entry for that object, if any.

  Directories may be observed and modified like objects. For purposes of access control, the directory for a protection level is assigned that protection level.

Operations

Accesses to any of the three components of objects, or to directories, are carried out by operations. There are eight classes of elementary operations, according to the two access modes and the four kinds of entities that can be accessed. Elementary operations may be requested in any sequence. There is no notion of current access state; each requested elementary operation is tested against the security policy.

Axioms

The first two properties, stated in terms of "objects", apply to each component of an object and also to directories.

Simple Protection Property: A subject may observe an object only if its current protection level dominates that of the object.

*'-Property: A subject may modify an object only if its protection level is dominated by that of the object.

Tranquility Principle: The protection levels of active objects will not change during normal operation. (An object is active if its identifier occurs in a directory.)

Discretionary Protection: A subject can access an object only if that access mode for that subject occurs in the permission matrix of that object.

Remarks

The assignment of a protection level to a relation as a whole may be contrasted with the SDC approach of a per-field assignment. Relations with fields of different levels can be simulated with multiple relations having the same primary key.

MAC MULTILEVEL SECURE DBMS SECURITY MODEL

Sources

Billy G. Claybrook and Harvey I. Epstein, "A Security Model for a Multilevel Secure Database Management System", The Mitre Corporation, Bedford, Massachusetts, M82-12, February 1982.

Overview

The database architecture underlying the security model and the model itself, are formally specified using the data abstraction approach. The database management system built on this model is intended to run on a trusted operating system. There is flexibility to enforce a number of different security policies; also the model is not specified with respect to any specific database data model.

Entities

- SUBJECTS: People, devices, processes, etc.

- OBJECTS: Data stored in the database and referred to as data granules.

- SUBJECT ATTRIBUTES:

  User Identifier: Identifies on whose behalf a request is made.

  Clearance: Consisting of a classification and category set (cf. glossary). The intention of the model was that there only be an ordered set of classifications, not necessarily only the four given in the definition in the glossary.

- OBJECT ATTRIBUTES:

  Data Content: One or more data elements. This is the data to be protected in the secure database. In a relational database, if the data granule is a relation then a data element is a tuple. If the data granule is a tuple then an element within the tuple is a data element.

  Classification: As defined in glossary.

  Category set: As defined in glossary.

Need-to-Know List: The set of all need-to-know elements, one for each possible type of access. A need-to-know element is a set of all user identifiers associated with a particular access type or mode. The access modes are as follows:

read: Allows user to read data.

modify: Allows user to alter data; this kind of access is not granted unless read access is also granted.

delete: Allows data to be removed from the database without being replaced.

Granule Identifier: A conceptualization used for specification purposes to uniquely identify data granules. It is not necessarily stored in the database with the data granule, but is either a tuple identifier (tid) in relational systems with granularity the tuple; or, equivalent to a data base key in CODASYL based systems with granularity the record. In practice, the former are determined by searching indices and the tid is then used to access the desired tuple. The latter are system generated values that uniquely identify records in the database and are thought of as stored record addresses.

Operations

As in the Bell-LaPadula model, each operation has certain preconditions involving the security attributes of the relevant subjects and objects. A violation is signalled when a precondition fails.

read: The contents of a data granule are returned.

delete: Delete removes the granule of a given identifier from the database.

N.B. The description of this operation in the source says that a user can delete only what he has read, write and delete access to; however, only delete access is checked in the formal specification of this operation on pages 16-22 in the document.

data_modify: This operation allows the user to modify the contents of a data granule.

data_move_modify: This operation copies one data granule into another data granule. This is an operation in which data is moved automatically from one part of the database to another without a user reviewing the data content. The data granule is overwritten and will receive the security tag of the overwriting data granule.

add: When new data enters the database, the data is tagged with its minimum access information supplied by the user or determined by the database management system. The user who first stores the data is allowed to retrieve or modify that data in case there are inputting errors. Add does not permit a user to write above his classification, i.e., the user must have read and write access to the set of data added.

change_class: To change classification. A user cannot upgrade classification of a data granule above his clearance level.

change_category: To change category set. A user cannot change the category set of a data granule unless he has all the compartments in the new category set.

change_ntk: To change the need-to-know list. A user cannot modify a need-to-know list unless he is on the read access list of that data granule.

Remark

This model specifies how access to objects in the database will be controlled based on the operations which can access those objects. This can be done by placing a "lock" on each operation. Each lock will require a "key" to enable a subject to use the operation. The key may be either an explicit or implicit key. Without the key for the lock, the requested function will be denied.

An explicit key may be a password or a coded magnetic strip required each time the function is invoked. Implicit keys are lists associated with each user indicating those operations the user is authorized to invoke. The system security officer modifies the user's list of authorized operations.

Keys are used as additional parameters in a user's request. If the required key is missing or incorrect, no database activity will be allowed and a security violation will be reported. System security officers handle all violations.

SDC SECURE DATA MANAGEMENT SYSTEM

Sources

Thomas H. Hinke, Marvin Schaefer, "Secure Data Management System", RADC-TR-75-266, System Development Corporation, Santa Monica, Ca., November 1975.

Overview

The SDC report describes a data management system that is to operate on a secure MULTICS operating system, using the existing directory structure and security enforcement mechanisms. The data base is assumed to be relational, consisting of a set of entries divided into fields. The model summary here blends the basic data access model, which merely restates the fundamental DoD requirements, with theoretical considerations stated later in the report.

Entities

- SUBJECTS: Users. Each user has a clearance, which is a classification.

- OBJECTS: An object is a field of an entry. Each field has a classification, independent of the entry.

- CLASSIFICATION: Consists of a military classification, a category, and a need-to-know list.

  military classification: A classification as defined in the glossary.

  category: Set of compartments; this model provides for sixteen compartments.

  need-to-know list: A set of users for each access mode. The possible access modes are read and write; other modes are interpreted as kinds of read and write for security purposes.

Axioms

A user may read an object if and only if the classification of the object is dominated by the clearance of the user, and the user belongs to the need-to-know read access list. "Domination" is determined in the usual way, ignoring the need-to-know lists.

A user may write into an object if and only if the classification of the object is precisely equal to the clearance of the user, ignoring the need-to-know list, and the user belongs to the need-to-know write access list.

Write access implies read access.

No field of an entry in a relation may receive a value prior to the assignment of values to every primary key field to that entry.

No field in the primary key of a relation may be modified.

The fields involved in the primary key to a relation must be classified at a level such that they are dominated by the classification of every remaining field in the relation.

All of the fields comprising the primary key to a relation must have identical classifications.

No subject may update a non-key field if the agent has the capability of reading any field whose classification strictly dominates the classification of the field. Updating fields based upon the contents of other fields is an asymmetric process with respect to field classification.

The deletion of an entry must be able to be accomplished by writing at only one level of classification.

NRL MODEL FOR SECURE MILITARY MESSAGE SYSTEMS (SMMS)

Source

C.E. Landwehr, C.L. Heitmeyer, and J. McLean, "A Security Model for Military Message Systems," NRL Report 8806, Naval Research Laboratory, Washington, D.C., May 31, 1984.

J. McLean, C.E. Landwehr, and C.L. Heitmeyer, "A Formal Statement of the MMS Security Model," Proc. of the 1984 Symposium on Security and Privacy, IEEE Catalog No. 84CH2013-1, pp. 188-194.

Overview

This model is for a family of systems sharing certain message characteristics, and differing from one another in the particular set of operations available to users.

Entities

- USER: A person authorized to use the MMS. A user has a clearance, userID, and a role. A clearance is a classification.

  A userID is a character string uniquely identifying the user, needed for logging in.

  A role denotes the job the user is performing, e.g., downgrader, releaser, distributor etc. A user can change a role during a session only if authorized for a given role. Each role has the ability to perform certain functions.

- ENTITY: Object or container.

  An object is the smallest unit of information in the system to which a classification is attached.

  A container has a classification and may contain objects, each with its own classification, and/or other containers; e.g., message files and messages are containers, some fields of a message are containers and devices will usually be containers.

  Entities have unique identifiers, used for direct references to them. An entity in a container may also be referenced indirectly with a sequence of names beginning with the unique identifier of the outermost container, giving

the location of the entity in a relative fashion.

Container Clearance Required (CCR): An attribute of some containers. For these containers, a user must be cleared at least to the container classification to have access to any of the entities in it.

Message attributes: A message will be a container which includes the following information: To, From, Date-Time-Group, Subject, and Text fields plus other fields such as Drafter and Releaser.

- CLASSIFICATION: A sensitivity level and a set of compartments. A sensitivity level is a classification as defined in the Glossary; a compartment is as defined in the Glossary.

- ACCESS SET: A set of authorizations that is associated with an entity. An authorization permits a specified subject, or any subject with a specified role, to use that object as a specified argument of a specified operation.

Axioms

Security Assumptions:

The System Security Officer (SSO) is assumed to assign clearances, device classifications and roles properly.

The user is assumed to enter the correct classification when composing, editing, or reclassifying information.

Within a classification, the user is assumed to address messages and to define access sets for entities he creates so that only users with a valid need-to-know can view the information.

The user is assumed to control properly information extracted from containers marked CCR (i.e., to exercise discretion in moving any information to entities that may not be marked CCR).

Security Assertions: The following statements hold for a multilevel secure MMS:

Authorization: A user invokes an operation on an entity if only if authorized by its access set.

Classification Hierarchy: The classification of any container is always at least as high as each of the classifications of the entities it contains.

Changes to Objects:	Information removed from an object inherits the classification of that object. Information inserted into an object must not be classified at a level above the classification of that object.
Viewing:	A user can only view (on some output device) an entity with a classification less than or equal to the user's clearance and the classification of the output device.
Access to CCR Entities:	CCR restrictions must be obeyed (see above).
Translating Indirect References:	A user can obtain the ID for an entity that he has referred to indirectly only if he is authorized to view that entity via that reference.
Labeling	Any entity viewed by a user must be labelled with its classification.
Setting Clearances, Role Sets, and Device Levels:	Only a user with the role of System Security Officer can set these.
Downgrading:	No classification marking can be downgraded except by a user with the role of downgrader who has invoked a downgrade operation.
Releasing:	No draft message can be released except by a user with the role of releaser.

DODIIS/DNSIX

Sources

L.J. LaPadula, "DoDIIS Security Protection for Information Exchange: Requirements," The MITRE Corporation, Bedford, Massachusetts, MTR-8756, Vol. I, August 1982.

L.J. LaPadula, "DoDIIS Security Protection for Information Exchange: Top Level Design Specification," The MITRE Corporation, Bedford, Massachusetts, MTR-8756, Vol. 2, November 1982.

Overview

The referenced documents do not explicitly describe a model, but rather a set of requirements and a high-level functional specification that answers those requirements. However, those aspects of the implied model that are apparent are described below.

The purpose of DoDIIS security protection is to allow a user at one node to access protected resources located remotely at another node. This requires that access control at each host must be augmented by an additional system called DNSIX (DoDIIS Network Security for information eXchange) to handle remote accesses. Note that a remote access involves activity by DNSIX systems at both the local and remote nodes.

Entities

- SUBJECT: A composite entity which includes a local user and session at a terminal, or a remote user and session at a host ("host" is synonymous with "node").

- OBJECTS: Datagrams (ingoing and outgoing) and protected resources.

- ATTRIBUTES: All of the above entities have an associated security level. Security levels are defined as in the Bell-LaPadula model.

  Each protected resource has the equivalent of an access control list, giving the access privileges of each user. The possible access modes are read, write, and execute.

A datagram has a source node and a destination node, and contains either data or an operation request.

- ACCESS STATE

The current state of a DNSIX system consists of a set of sessions. Each local session is associated with a local user and a terminal. Each remote session is associated with a remote user and a node. The DNSIX at the remote node must have a corresponding local session with the same security level and user. A DNSIX system also has an audit trail file. The handling of every operation request has the side effect of creating an audit trail record specifying the time, source, and disposition of the request.

Operations

Identification	Each operation request is accompanied by the identification of the session responsible for it.
Local vs. Remote	Any operation request to DNSIX may come either from a local user or (via a datagram) from a remote user. Requests to remote hosts are generated from local requests to create the appropriate datagrams.
Session Initiation	Operations exist to request the creation and deletion of local or remote sessions.
Access	The model does not record "current access" as part of the state; instead, accesses are controlled on a transaction or operation basis. Hence, there are operations for exercising access in read, write, and/or execute mode to protected objects.

Axioms

Login:	A local session level is dominated by the associated user and terminal levels.
Node Association:	A remote session level is dominated by the remote host level.
Mandatory Access:	A request for access to a protected resource by a user in a session is permitted only if the resource level is dominated by the session level

Discretionary Access:	A request for access to a protected resource by a user in a session is permitted only if the requested mode is contained in the user's privilege.
Security Label (Outgoing):	The level of an outgoing datagram is dominated by the local host level.
Security Label (Incoming):	The level of an incoming datagram must be dominated by the remote (source) host level.

ACCAT GUARD

Sources

J. Keeton-Williams, S.R. Ames, Jr., B.A. Hartman, and R.C. Tyler, "Verification of the ACCAT-Guard Downgrade Trusted Process, Volume I: Overview and Major Results," The MITRE Corporation, MTR-8463, Vol. 1, September 1981.

J. Keeton-Williams, B.A. Hartman, J. Abbas, and R.C. Tyler, "Verification of the ACCAT Guard Downgrade Trusted Process, Volume III: Specification and Proof", The MITRE Corporation, MTR-8463, Vol. 3, January 1982.

Overview

The ACCAT Guard system connects two networks which have different security classifications. The higher-level network is the Navy Ocean Surveillance Information Systems (OSIS) Baseline; the lower-level one is the Navy's ACCAT (Advanced Command Control Architectural Testbed) system. lower-level one is the ARPANET. process, called the DGTP (DownGrade Trusted Process), implemented on top of a kernel (KSOS-11). The DGTP uses kernel commands to accept file downgrading requests from the higher-level network, display the associated file at a terminal for the watch officer, accept the watch officer decision (accept or reject), and (if accepted) send the file to the lower level network. A significant aspect of the model and subsequent verification effort is that the watch officer terminal was explicitly included as part of the DGTP system and the verification effort thus required analysis of the properties of the terminal.

The model given below is a top-level, or security, specification of the DGTP, called S0 in the references. The role of the kernel is invisible at this level.

Entities

The state consists of four sequences, each of which represents the history of transactions which have occurred at one of four points in the DGTP interface:

1. files_viewed: files from the DGTP system to the watch officer;

2. swo_decisions: decisions from the watch officer to the DGTP system;

3. msgs_to_low: messages from the DGTP to the lower-level network (passed on with some modification from the higher-level network);

4. files_to_low: file notes from the DGTP to the lower-level network.

Inputs from the higher-level network to the DGTP are not recorded as such in the system state, but exist implicitly as a way of supplying a file identifier, a message, and a file for DGTP action. A message is either a control message or not. If not, it contains a file identifier. A file note contains a file identifier and a file.

Axioms

A sequence of system states is secure if and only if it is a DGTP_EVOLUTION, defined as a concatenation of QUALIFIED_DOWNGRADEs and NONPRIVILEGED transitions.

A NONPRIVILEGED state transition is one in which there is no change in either the files_to_low or msgs_to_low histories.

A QUALIFIED_DOWNGRADE state sequence consists of a DISPLAY followed first by a SWO_REPLY and then by a DOWNGRADE.

A DISPLAY is a state transition in which an input file is appended to the files_viewed history.

A SWO_REPLY is a state transition in which a "swo_accept" decision is appended to the swo_decisions history.

A DOWNGRADE is a state transition in which the msgs_to_low and, usually, the files_to_low histories are augmented. The new msg_to-low entry is the input message with its identifier field replaced by the input identifier. The new files_to_low entry is a file note consisting of the input identifier and the input file. In the case of a control message, the file identifier does not exist and the files_to_low history remains unchanged when passed to the lower-level network.

LSI GUARD

Source

S. Stahl and J.G. Keeton-Williams, "LSI Guard Security Specification," MTR-8451, Revision 1, The MITRE Corporation, Bedford, MA, August 1982.

Overview

An LSI Guard system, like an ACCAT Guard, stands between two systems at different security levels, denoted "low" and "high". Messages go from one of these two systems to a Guard user, who may then request that it be sent (from that user) to the other side.

The model is written in the same formal style as the ACCAT Guard model, except that the state components are individual messages rather than message histories.

Entities

The LSI Guard state consists of a flow control matrix, a communication output, and a downgrade message.

A flow control matrix is indexed by sender and receiver; senders and receivers are either users or one of the special participants "low", "high", or "fcm" (representing the flow control matrix itself). The matrix entry for a given sender and receiver is a set of permitted messages.

A communication consists of a sender, receiver, and a message. Requests to the system from a sender are in the form of communications, and cause state transitions. The appearance of a communication output in the new state indicates that the message is actually transmitted to the receiver.

A message has a type label and a content.

A secure start-up state is empty except that there exists a user authorized to send any "tsso" (terminal system security officer) message.

Axioms

A state sequence is secure if it begins with a secure start-up state, and every transition in it is a secure event. A transition is secure only if the input communication is authorized in the flow control matrix.

A secure event either enters an authorized communication output into the new state, updates the flow control matrix, or enters a message into the downgrade message component, where it can be viewed for further procesing.

Flow control updates are caused by input communications addressed to "fcm". Several particular types of updates are specified. The model identifies various user "privileges" which summarize classes of entries in the flow control matrix. These permit users, for example, to connect with the low and high sides, or modify the flow control matrix to allow emergency direct connection between the two sides.

MESSAGE FLOW MODULATOR

Source

D.I. Good, A.E. Siebert, L.M. Smith, "Message flow modulator final report," Technical Report 34, Institute for Computing Science, The University of Texas at Austin, Austin, Texas, December 1982.

Overview

The message flow modulator is a trusted program, implemented in Gypsy, designed to monitor the flow of security sensitive message traffic from the U.S. Naval Ocean Surveillance Information System.

Entities

The message flow modulator (MFM) receives messages from a source and forwards passed messages to a destination and rejected messages to a log.

A message is a sequence of ASCII characters beginning with "ZCZC" and ending with "NNNN".

The MFM has a table of patterns. A pattern is a sequence of characters representing a set of character sequences matching it according to certain wild-card conventions. Patterns are used to specify key words that indicate that a message is security sensitive and should be rejected.

Axioms

A message is passed if and only if it does not contain any substrings matching any of the patterns in the table.

SRI MODEL OF MULTILEVEL SECURITY

Source

R.J. Feiertag, K.N. Levitt, and L. Robinson, "Proving Multilevel Security of a System Design," Proc. Sixth ACM Symp. on Operating Systems Principles (November 1977), 57-65.

Overview

This model is the oldest of several related models taking a particular, very abstract approach to defining security, as the invisibility of higher-level users to lower-level ones.

Entities

A secure system is a black box, or sequential abstract machine, whose inputs and outputs have security levels associated with them. Inputs are called operations and are in the form of function invocations, consisting of a function name and arguments for the function.

Security levels are partially ordered, larger elements representing greater sensitivity. The same security level format is used as in the Bell-LaPadula Model.

- SUBJECTS: Subjects are not explicitly represented in the model. They exist implicitly as the source of inputs and the destination of outputs.

- OBJECTS: There are actually two models: a general model, which does not need the concept of an object; and a restricted model, which does. In the restricted model, the internal state of the system is assumed to be comprised in the assignment of values to a set of state variables, which play the role of objects. Each state variable has a constant security level.

Axioms

In the general model, there is only one axiom. This states that after a sequence of operations, the last output depends only on those operations of lesser or equal security level.

In the restricted version of the model, there are three axioms given which, together, are stronger than the axiom of the general model:

(a) The output of an operation depends only on the values of state variables of lesser or equal security level.

(b) The new value of a state variable (after an operation) depends only on the values of state variables of lesser or equal security level.

(c) The value of a state variable can be changed only by an operation of lesser or equal security level.

Remarks

The reason for introducing the restricted model, with state variables and stronger axioms, is that the three axioms are easier to check when the system is specified formally and nonprocedurally in a language like SPECIAL.

The model covers non-discretionary security and non-disclosure properties only. Because of the recognition of state variables as objects, an analysis technique based on this model can detect covert storage channels.

RUSHBY'S SEPARABILITY MODEL

Source

J.M. Rushby, "Proof of Separability: A Verification Technique for a Class of Security Kernels," International Symposium on Programming, Lecture Notes in Computer Science 137, Springer-Verlag, New York, 1982, pp. 352-367.

Overview

The "separability" policy defined by this model is a policy of isolation between users, so that the behavior of the shared system is indistinguishable to each user from that which could be provided by a machine dedicated to that one user.

Entities

A system is modeled as an abstract machine with states, inputs, outputs, and functions to determine the next state and output from a given state.

Inputs and outputs are vectors with a component for each user.

Machine transitions occur in pairs: an optional input transition followed by an autonomous state-state transition. Corresponding to the sequence of states is the associated sequence of outputs.

Axioms

A machine is secure if each user's sequence of output components is independent of the input components from other users, in the sense that another sequence of inputs having the same components from this user would result in an output sequence with the same components for this user.

This definition excludes timing channels, since the machine is synchronous, and denial of service. A weaker definition is given that does not count the number of consecutive transitions a particular output component appears, and ignoring the premature termination of an output sequence. The weaker definition is viewed as more realistic, since the exclusion of timing channels and denial of service is beyond current security kernel technology.

Remark

The separability policy can be applied to some systems where limited inter-user communication is possible, by applying

the policy to a slightly modified system where the desired communication paths are, for verification purposes, cut.

GOGUEN-MESEGUER NON-INTERFERENCE MODEL

Sources

J.A. Goguen and J. Meseguer, "Security Policies and Security Models," Proc. of the 1982 Symposium on Security and Privacy, IEEE Catalog No. 82CH1753-3, pp. 11-20.

J.A. Goguen and J. Meseguer, "Unwinding and inference control," Proc. of the 1984 Symposium on Security and Privacy, IEEE Catalog No. 84CH2013-1, pp. 75-86.

Overview

The non-interference notion expressed in this model is the most general idea underlying the models in the SRI heritage.

Entities

A system has states, users, and two types of commands: state-changing commands and output commands. The next state and output depend on a user (the user initiating the state change or output) as well as on a state and command. In the earlier form of the model, a state was partitioned into a capability table component and a remainder, and state-changing commands affected only one of the two components at a time.

Axioms

The model can support any policy expressible as a set of non-interference assertions. A non-interference assertion specifies that state changes initiated by a given user cannot affect the outputs received by another given user. Specifically, the second user would receive the same outputs if the first user's state changes (since some initial state) had not occurred. The definition generalizes to sets of users and can specify particular sets of commands.

A multilevel security policy can be expressed by assigning security levels to users and requiring that a user not interfere with other users having lower or incomparable security levels. This is essentially the same policy stated in the SRI model.

Remarks

The model can also express Rushby's separability policy, and a variation on non-interference can be used to permit limited communication between users.

SECTION 3

USING MODELS TO DESIGN MODELS

VALID INTERPRETATIONS OF MODELS

Security implications of an abstract model are passed on to a more concrete model when the latter is a "valid interpretation" of the former. This means that the axioms satisfied by the abstract one are also satisfied in the concrete one, and hence all theorems about the abstract one are also true for the concrete one. In particular, the security axioms and their consequences will be carried over.

A good introduction to the concept of valid interpretation of models, in a computer security context, may be found in a CWRU report [WAL74]. Enough of the essential ideas will be summarized here to allow some observations to be made.

A model is viewed as consisting of a collection of entities and a collection of relations among those entities that satisfy some axioms. Typical entities in a security model are subjects, objects, security levels, access modes, and states.

To set up a correspondence between two models, one needs a mapping of entities and relations. A more concrete model may rename the entities, and also may refine them into subclasses. For example, both users and processes may play the part of subjects. The mapping of entities between the models will then say that a user is mapped to a subject, and so is a process. Note that only the concepts of "user" and "subject" are linked by this, not specific users or subjects.

A mapping of relations is needed to translate axioms in the abstract model into the notation of the concrete one. We will go into this process in some detail because the way mappings work is a fundamental and important idea.

Suppose, for example, that one axiom of an abstract model is this form of the simple security condition:

> If a subject has read-access to an object, in any state, the level of the subject dominates the level of the object in that state.

Now we want to reinterpret this axiom for a more concrete model in which users and processes are mapped to subjects, files to objects, and levels, access modes and states remain the same. The axiom starts out with a reference to a subject having some kind of access to an object in a state. This is evidently a relation joining a subject, an object, an access mode, and a state. In order to proceed we will have to be more formal. Let us assume that this relation is denoted by:

b(s,o,m,q).

(Why "b"? The model designers pick the names of the relations - you are stuck with what they chose. This is not an atypical example, since Bell-LaPadula has a very similar relation called "b".)

The simple security condition in the abstract model is, in this more formal notation,

If b(s,o,m,q) and m = "read" then ...

This axiom must be restated in the terminology of the concrete model. The concrete model will not (not necessarily, anyway) have a relation named "b", so we must look at the mapping to tell us about how to determine what kind of access users or processes have to files. Suppose there is a relation

Access(p,f,m,q)

among processes, files, access modes, and states in the concrete model. (Concrete models tend to have more mnemonic relation names.) If it is intended that this relation represents the same thing as "b" in the abstract model, the mapping will say so, in a form like:

If s is a process,
b(s,f,m,q) iff Access(s,f,m,q).

This statement effectively defines the relation "b" for the concrete model in the process-subject case. Another statement would be needed to cover the user-subject case; note that a different relation, instead of "Access", might be used in that case. Also, the right-hand sides of mapping definitions need not be as simple as this. A logical expression involving two or more concrete relations may be needed.

With the mapping of "b" to "Access" set up, we can see that the simple security condition in the concrete model will begin:

If s is a process and Access(s,f,m,q),
or s is a user and ...

This translated axiom, derived from the abstract model but stated in the context of the concrete model, may or may not be true. The concrete model has its own axioms, and this need not be one of them. If we want to prove that the concrete model is a valid interpretation of the abstract one, we must show that this translated axiom, and any others, are theorems provable from the concrete model's own axioms.

MODELS WITH OPERATIONS

It was not hard to see how axioms are translated into more concrete forms when they only involve a single state. It is less obvious what to do when the abstract model or the concrete one, or both, have axioms restricting state changes. Most models have some restriction on state changes, if only to say that subject security levels or object security levels do not change. Models with operation specifications, like the rules in Bell-LaPadula, impose severe restrictions on state changes.

From a logical point of view, states do not "change"; instead, states are entities that are related to other states by standing in the relationship of "succession". This relation can be mapped, like others, when checking for valid interpretations.

It is, of course, much easier to ascertain that this process can be carried out in principle than it is to perform it for a particular pair of models.

COMBINING MODELS

The process of combining the features of two models can be viewed as finding a model that is a valid interpretation of both of the two models. The first step in the combination is to find a single terminology for the entities and access modes, etc., that can be mapped conveniently to both models. The combination is then checked to see that it satisfies the concrete versions of the axioms of both of the original models.

APPLYING INFORMATION FLOW MODELS

Some very abstract models, such as the ones surveyed in this report in the "General Models" category, are concerned with transfer of information in a more general sense than copying data from one object to another. They were originally motivated by the observation that covert channels for compromising information exist

in systems that were secure according to some access control model. Kernel access-denial responses, for example, carry information that might be modulated by Trojan horses, invisibly to access control models.

As it happens, the precision inherent in information flow models also makes it possible to show that certain systems are secure despite violations of an access control model. The best example is a trusted process that has read access to high level files and write access to low level files, but avoids compromises because its algorithms do not mix the information.

Information flow models are analysis rather than synthesis tools; they are no help in suggesting the functional characteristics of a model or design. They are most useful for models having operation specifications. In fact, they have been applied in the past primarily to analyze formal system specifications rather than models.

Much can be gained by applying flow analysis techniques to concrete models having operation specifications. The flaw mentioned in Section 1 in the Bell-LaPadula model, for example, involving the change-current-security-level rule, would have been discovered by an information flow analysis. Incidentally, one practical way of applying existing tools to analyze a model this way is to translate the model into a specification language for which a flow analysis tool is available.

SECTION 4

A MODEL FOR A1 POLICY

Recently, the question of the proper function and design of a model has come under intense scrutiny, largely because of the publication of the Department of Defense Trusted Computer System Evaluation Criteria [CSC83]. A formal model of the applicable DoD security policy is required for systems to be rated in the higher protection classes.

It has been recognized that no one model will serve the needs of all applications. Nevertheless, it is felt to be beneficial to have a model that addresses the policy stated for A1 systems, the highest class included in the Criteria. It is anticipated that such a model could be used in two ways. It could be the model of the security policy supported by a proposed TCB (Trusted Computing Base), as required in the Criteria. It could also serve as a "kernel" around which more elaborate models can be built.

A model with those objectives is necessarily constrained in style and applicability. First, its subject matter and content are constrained to express the A1 security policy as stated in the Criteria. Hence, it deals with subjects, objects, security classifications and categories, and must include a particular restriction on the ability of subjects to read or write objects on the basis of their respective security levels.

Like the Criteria document itself, the model will be limited in application to general purpose operating systems. While it is possible to build a secure message system, data management system, guard system, or network switch on top of a secure general purpose operating system, one would expect the security policy in each case to have various unique features. In some cases they could be added to the A1 model in the form of a superstructure or concrete interpretation, but in other cases it may be more practical to construct a different model altogether.

The Criteria document suggests that the Bell-LaPadula model would be acceptable as a formal model. It is doubtful, however, that it is the best choice for a model for a new TCB to be submitted for A1 certification. One reason is that it is unnecessarily restrictive - it includes specific "rules" for system functions, which may be incompatible with the desired TCB functions, and it includes a Multics-directory-like object hierarchy. The set of rules "...is in no sense unique, but has been specifically tailored

for use with a Multics-based information system design" [BLP75, p. 19]. Another reason is the covert channel in that model, discussed in Section 1, together with the possibility that other such channels exist.

In designing a new model to address A1 security policy, the initial objective was to create a "minimum model", one that covered the Criteria requirements and nothing more. Consequently, while the suggested A1 model below resembles the Bell-LaPadula model in having subjects, objects, accesses, and a form of the *-property, it does not have rules for specific operations or an object hierarchy.

As other authors have discovered, however, there is an overwhelming temptation for a model designer to add new features in response to perceived deficiencies in other models. The model described below has two innovations. One expands the treatment of trusted subjects in a way intended to be more flexible and effective; the other incorporates discretionary security into the mandatory security level.

Trustedness of subjects is dissected into a collection of separate privileges which must be inherited by the subject from the objects to which it has execute access. Objects possessing such privileges are required to have high integrity, enforced by a component of the security level.

Discretionary security is handled by adding user-list components into the security level. Since security levels can be changed only by privileged software, the effect is to prevent individual access control from being subverted by Trojan horses.

INFORMAL STRUCTURAL DESCRIPTION

System State

The system state consists of a set of subjects, a set of objects, and some functions defining their current status. Each subject and each object has a security level and a (possibly empty) set of privileges. Associated with each subject is a set of objects to which it has read access, a set to which it has write access, and another set to which it has execute access.

A security level has the following components: classification, category set, integrity class, integrity category set, distribution list, and contribution list. The partial ordering "dominates" of security levels is based on the ordering of each of the components. The third through fifth components are ordered inversely, i.e., a greater level has a smaller value in those components.

Transitions

A transition is a state change in response to a request from some subject, called the requestor. There will be security conditions defining restrictions on secure transitions as well as on secure states.

Some transitions create subjects or objects. Mathematical entities are never really "created", of course; this just means that the set of subjects or objects associated with the next state is larger. Subjects or objects can also be deleted. Note that, since every "existent" subject and object has a security level and other attributes, any creations or deletions imply a change in those components of the state as well.

SECURITY CONDITIONS

Several of the security conditions given below are waived for subjects having an appropriate privilege; those conditions are starred (*). Subjects inherit their privileges from the objects they execute. Privileged subjects and objects must have a particular integrity category, called "Trusted", in order that their trustworthiness may be preserved.

A subject can grant a privilege p to an object only if it has a special privilege (Create-p) to do so. In an effort to control the propagation of privileges, we require that no privilege can create itself, either directly or indirectly. (To ensure this, define a function "Create" such that Create(p) = Create-p, satisfying the restriction that, for any set A of privileges, Create(A) cannot be a subset of A.)

Secure State Conditions

The Read and Write conditions below are derived from [CSC83, section 4.1.1.4, p. 45].

* Read: The level of a subject dominates the level of any object to which it currently has read or execute access.

* Write: The level of a subject is dominated by the level of any object to which it has write access.

Privilege: The privilege set of a subject is included in the privilege set of any object to which it has execute access.

Trust: If a subject or object has any privilege, The integrity category component of its security level includes the Trusted category.

Secure Transition Conditions

Transition conditions are waived only when the requestor (rather than any other subject mentioned) has the appropriate privilege.

* Tranquility: The security level of a subject or object does not change. Note that a change has different effects on different components of the security level. Separate privileges may be required for changes in different components of the security level.

* Creation: The security level of a new subject or object dominates that of the requestor.

* Access change: Only the accesses of the requestor can be changed.

Privilege change: A privilege p can be entered into the privilege set of an object only if the requestor has the privilege Create-p.

SECURITY LEVEL COMPONENTS AND ORDERING

A security level has six components, the first two having to do with information sensitivity, the next two with integrity, and the last two with individual access control. A security level dominates another if its sensitivity and contribution list components are greater (or equal) and its other components are less (or equal). The general principle is that a higher security level implies a greater restriction on access.

The partial ordering for each component is given below with the component description.

Classification

Usually one of the following: Unclassified, Confidential, Secret, and Top Secret. However, eight classifications are required for some National Security applications, according to guidance in [CSC83]. The classifications given above are linearly ordered, Unclassified being the least and Top Secret the greatest.

Category Set

Individual categories vary with the community, but a given system should support at least 29 categories to represent document compartment markings, according to guidance in [CSC83]. Furthermore, some additional categories may be needed to represent dissemination controls and other distribution limiters. Category sets are ordered by set inclusion, the empty set being the least and the set of all categories the greatest.

Integrity Class

Because the integrity class is an inversely ordered component, a subject can only read from a higher or equal-integrity object and write into a lower or equal-integrity object, so copying and computational operations cannot increase integrity. This form of integrity control, using the Read and Write conditions on a "dual" or inversely ordered integrity classification, comes from Biba's "strict integrity" model [BIB77]. Incorporating an integrity component into the security level was done first in the I.P. Sharp Protected Data Management System Model [GRO76].

There is some support for the idea that security classifications also carry a connotation of integrity. This idea can be implemented by having integrity classes Unclassified through Top Secret, with the understanding that the integrity class is not necessarily equal to the security classification. Typically one would expect that the security classification dominates the integrity class.

Integrity Category Set

Integrity category sets are ordered by set inclusion just like (sensitivity) category sets. Because it is an inversely ordered component, copying and transformation operations can only reduce the set of integrity categories.

In this model, there is a "Trusted" integrity category, which is intended to be used for objects containing software that will be executed by privileged subjects.

It might be asked why the individual privileges could not be implemented as integrity categories. The Read condition would then require that subject could not have a privilege unless the object to which it had execute access also had that privilege. The problem is that all the objects to which the subject had read access would have to have that privilege as well.

Distribution List

This is the set of names of users who are permitted to read an object. Distribution lists are ordered by inclusion, the empty set being the least and the set of all users being the greatest.

Because the distribution list is one of the inversely ordered components of the security level, it follows from the Read and Write conditions that an object can be copied or transformed only into another object with a smaller or equal distribution list. This prevents information from receiving a wider distribution than originally intended.

It may be surprising that a subject has a distribution list. The intent here is that the subject's distribution list represents a mode of operation during a temporary session, and it places an upper limit on the distribution of information it is currently handling.

There is no notion within this model of a particular user on whose behalf a subject is operating. In order for the distribution list to have the desired effect of limiting the users who can receive information, there is an assumption we have to make about how the system being modeled is interfaced with the outside world. We assume that an output device being operated by a user is an object (or more than one object); and its distribution list should include that user. This assumption would be included in the security requirements for a trusted login process. The role of users is explained further below in a separate subsection.

In view of the fact that the distribution list is part of the security level, and the security level cannot be decreased by an unprivileged or untrusted subject, there may be some question whether this mechanism satisfies the intent of "discretionary" security.

The guidance for discretionary security in [CSC83], which is extracted in turn from DoD regulations, mentions two points: access control on an individual basis, and need-to-know. The distribution list mechanism clearly qualifies on the first point. As far as need-to-know is concerned, one thing is certain: authorization of need-to-know cannot be left to a Trojan horse. We know that Trojan horses are a concern, because they were the rationale for introducing the *-property, which reappears in this model in the form of the Read and Write conditions. Only a trusted, specifically privileged process can be expected to reflect the intent of an appropriate user when the distribution list is expanded.

The difference between a mandatory label like classification and a discretionary one like the distribution list is that a specific system administrator or operator must be consulted to change the former, while the custodian or owner of the object has sufficient authority to change the latter. This security policy should be embodied in the specifications for the privileged software for each of those tasks. It is not embodied in the model because of the difficulty of capturing the intent of a user. In the model, a user could be identified as an owner of each object, but this was not done because there is no formal axiom that explains the meaning of the relationship.

The term "discretionary" is confusing here, because changes in the security level are normally the province of "mandatory" or "non-discretionary" control. Perhaps the distribution list mechanism should be referred to as "individual" control instead.

Another possible objection is the apparent need to specify users one-by-one on the distribution list; The Al security policy calls for the ability to specify access for whole groups at a time. The apparent discrepancy is due merely to the level of abstraction of the model. A system implementing this model can specify sets of users symbolically in any desired way, as long as it is clear which individual users are included.

Contribution List

The contribution list is a set of users, like the distribution list, ordered by set inclusion. It is intended to implement individual control on write access. As in the case of the distribution list, we need an assumption about the external interface: the name of any user operating an input device must be on the contribution list of the object representing that device.

The Read and Write conditions imply that the contribution list of any object contains all users who may have influenced it, or will be permitted to influence it in the future.

INPUT, OUTPUT, AND USERS

Users were previously mentioned in the context of individual access control as elements of distribution lists and contribution lists. They have a role in modeling the external interfaces to a system, to explain the source of input and the destination of output.

When a model like this is considered in a larger context of network security, it becomes important to have a more precise notion how input and output are handled. For this reason, we will now give more detail on how users may be incorporated formally into the model.

Let us say that users are actually special subjects. They are exceptional because they cannot have execute accesses, and because they cannot have both read and write accesses. This split between a user as an output sink and as an input source reflects the lack of a deterministic circuit between the outputs to a user and its subsequent inputs to the system. A human operator would be modeled as a pair of users (eyes and hands respectively). A full duplex network connection is a pair of simplex connections.

Naturally, the distribution list of an output user consists only of that user; the contribution list of an input user is also just that user. The distribution list of an input user and the contribution list of an output user are, by convention, all-inclusive, so as not to interfere with users' source and sink roles.

Users are also exceptional in that they cannot directly request changes in the system state, such as access or level changes. Requests of this kind do not really come from users; they come from processes running software that has interpreted the user's keystrokes. This is why we cannot say, for example, that any user, as a subject, has the privilege to change the individual access components of any object it owns.

CONCLUSIONS

The model described in this section is too complex to qualify as a "minimum A1 policy model", but it embodies some suggestions about what is needed in models and leads to some conclusions about what is still missing.

A minimum A1 policy model could be obtained from the one given, by leaving out all security conditions except Read, Write, and Tranquility; and dropping the integrity components of the security level. The result would still have an unusually restrictive interpretation of discretionary security. The mandatory control of individual access might, however, be just right for an A2 model.

SECTION 5

CONCLUSION

The procurement package for a new system includes security requirements. These requirements should be specified in enough detail to make it possible to select or construct an appropriate model or set of models. Having looked at a number of security models, it is possible to give at least some broad answers to the basic questions one faces at the outset of the modelling process.

1. Are there different types of security models?

There are a number of useful distinctions one can make among security models, though most of them are shades of gray rather than black/white dichotomies. First, models can be abstract or concrete. The purpose of abstract models is to investigate fundamental security issues, such as information flow, using mathematical techniques. The purpose of concrete models is to guide the design of a secure system, subsystem, or family of systems. Concrete models can be categorized according to the interfaces they present and the assumptions they make about other systems they must interface to. For example, does the model represent a whole system, like a secure network, or just a trusted process or a policy manager? Within an interface type, a model can be more or less adapted to a particular application, such as a DBMS, message system, or guard.

2. What kind of axioms should a model have?

All models have axioms giving conditions for security or consistency. Axioms may restrict: (I) the system state, (II) transitions from one state to another, (III) actions performed with data objects, or (IV) outputs at various interfaces. The system state of a model that includes a policy manager indicates the types of operations currently permitted to subjects on objects. Policy manager models deal with operations on data objects by categorizing them and using tokens, usually called access modes, to represent each category. There may be only a few access modes, such as read and write, or there may be a large variety of modes representing specific computations such as relational projection. Note that axioms of type III are possible only if the model includes an object manager.

3. Should a model give specifications for individual state-changing operations, like the Bell-LaPadula rules?

It is not necessary to limit the state-changing operations to a fixed list that are individually specified; one can, instead, attempt to give general conditions for secure state changes. The rules in the Bell-LaPadula model served two purposes: one was to provide more constructive design guidance, an objective which is now typically deferred to a separate formal specification phase; and the other was to disallow other operations which might satisfy the axioms, but have other problems like information flow leakage. Now that general techniques are available for testing for information flow leakage, it is unnecessarily heavy-handed for a model to limit operations to a specific list on those grounds.

Models which include the actions of trusted processes, such as guard models, have a specification-like format because their security requirements do not lend themselves to general axioms.

4. Should models be formal (in a mathematical or programming language) or in English?

Every model should have an English (or other suitable natural language) version to clarify its motivation, objectives, and concepts. Like system specifications, however, models have problems and security vulnerabilities that are not easy to pin down unless the model is expressed formally. Furthermore, verification techniques for system specifications can be applied only with respect to a formal version of a model.

5. How does one determine whether the model itself is "secure"? (i.e., adequately and correctly expresses a reasonable security policy?)

Models generally do not completely express all aspects of system security, but some tests are available to search for flaws in what is provided. One can check that a model is a valid interpretation of a previous model for part of its policy, and one can apply information flow analysis to look for covert channels. These techniques are tedious to apply, and are generally used only informally. Those who apply them informally, however, need to understand in principle how to perform them rigorously. Another, more common-sense, approach is to match the intended implications of the model against its actual logical consequences. The process of formalizing a verbal model often helps to expose inadequacies.

6. How does one decide whether to use an existing model or design a new one?

The first step is to identify the security-critical components that will be developed using a model-based methodology, so that one

can tell how many models are needed, what they interface with, and what assumptions they can or cannot make about other subsystems. One must know the role of each subsystem, e.g., kernel, DBMS, or guard.

By looking at the kinds of entities and attributes that appear in the models surveyed here, one can get an idea what information needs to be provided about the system users, the protected resources, and the kinds of information objects that pass between systems (messages, etc.). If the security levels in a particular application are more or less complex than the common classification-category set version, that will have to be specified.

When a review of known models with respect to their support of security policy and functional requirements indicates that no model is adequate, a new model can be created by combining or specializing existing models or combining features from several. There are pitfalls in this process - models can have mistakes or oversights built into them - but the same remark applies to the original model or models.

REFERENCES

[BIB77] K.J. Biba, "Integrity Considerations for Secure Computer Systems," ESD-TR-76-372, The MITRE Corporation, Bedford, MA, April 1977.

[BLP75] D.E. Bell and L.J. LaPadula, "Secure Computer System: Unified Exposition and Multics Interpretation," ESD-TR-75-306, The MITRE Corporation, Bedford, MA, July, 1975.

[CSC83] "Department of Defense Trusted System Evaluation Criteria," CSC-STD-001-83, 15 August 1983.

[FEI81] R.J. Feiertag, "A model of security policy for the integrated computer network (ICN)," SYTEK-TR-81001, SYTEK, Inc., Sunnyvale, CA, April 1981.

[GRO76] M.J. Grohn, "A Model of a Protected Data Management System," ESD-TR-76-289, I.P. Sharp Associates, Ltd., Ottawa, Canada, June, 1976.

[LAN81] C.E. Landwehr, "Formal Models for Computer Security," Computing Surveys, Vol. 13, No. 3, September 1981, pp. 247-278.

[WAL74] K.G. Walter, et al., "Modeling the security interface," Case Western Reserve University, Cleveland, OH, August 1974.

APPENDIX

GLOSSARY

A number of terms have a special meaning in the context of computer security modeling, which may differ from their usage in the DoD information security context. Furthermore, different models often disagree on the meanings to be attached to certain terms. An attempt has been made to give definitions reflecting the most common usage of these terms in the computer security community, although occasionally a preferred definition is given. Remarks and examples are often included with definitions.

Access: A relation between a subject and an object that is restricted by a security policy. Access implies the ability of the subject to perform some operation or class of operations on the object. Access can be in various modes, and it can be gotten and released. Typical modes of access are: read, write, execute, control, append, delete.

Aggregation: A circumstance in which a collection of information has a classification higher than any individual item.

Audit Trail: A record of reference monitor transactions intended to permit the identification of a user or program responsible for harmful activity.

Capability: An unforgeable ticket, in the possession of a process, authorizing a particular operation or mode of access to a specific object.

Category: Compartment.

Channel: A means for transferring information from one system entity to another.

Classification: A security label from a linearly ordered set, reflecting the sensitivity of information or the level of clearance of an individual. The usual classifications for DoD applications are UNCLASSIFIED, CLASSIFIED, SECRET, and TOP SECRET (although UNCLASSIFIED is not a "classification" in the DoD sense of the word). Eight classifications are required for some National security applications.

Compartment: A security label reflecting access restrictions on the basis of need-to-know. A compartment corresponds roughly to an interest group or topic area, for example: NATO, NUCLEAR. In a

computer system, compartments may also be used to represent dissemination controls, caveats, and other distribution limiters, e.g., NOFORN, FOUO, and NOCONTRACTOR.

Covert Channel: A channel that violates the intent of the security policy without violating its statement.

DBMS: Data Base Management System.

Discretionary Policy: A security policy controlling access of individual users to information, on the basis of need-to-know.

Dominates: A partial ordering on security levels. For example, SECRET dominates UNCLASSIFIED. It is a weak ordering in the sense that it includes equality, i.e., any level dominates itself.

Guard: A computer system acting as an information filter between two (or more) systems operating at different security levels.

Information Flow Model: A model for determining whether a given output of a system was (or was not) influenced by a given input.

Integrity Policy: A security policy offering protection against unauthorized modification or destruction of information.

Kernel: The implementation of a reference monitor; it controls accesses of processes to data objects. It includes a policy manager and an object manager.

Mandatory Policy: A security policy that restricts access on the basis of the security levels of subjects and objects.

Multilevel System: A system having users who are not authorized for all the information present in the system, where authorization is determined on the basis of security level.

Non-Discretionary Policy: Mandatory policy.

Object: A repository of information; a participant in an access relation.

Object Manager: A component of a secure system whose state determines whether a process can perform an operation on an object.

Policy Manager: A component of a secure system that receives and disposes of requests to change the current access state. Besides a decision response to the requestor, it has an interface with the object manager to enable approved access changes.

Reference Monitor: An entity that mediates accesses by subjects to objects according to a specified protection policy. A reference monitor must be:

1) complete: it is always invoked;

2) isolated: it is tamperproof;

3) provably correct: it is small enough to be subject to analysis and tests, the completeness of which can be assured.

Sanitization: To delete sensitive information from a file, or modify it, so as to lower the file's security level.

Security Level: A security label on system entities, referenced by some part of the security policy. Security levels are usually partially ordered (some pairs of levels may be incomparable). The partial ordering is a lattice if each pair of levels has a least upper bound and a greatest lower bound.

Security Policy: System requirements intended to prevent unauthorized disclosure, modification, or destruction of information. Denial of system service is also considered a security concern if it is due to malicious activity.

*-Property: (Pronounced star-property): A security policy restriction that prevents a subject with information at one security level from writing that information into an object that is not of an equal or higher level. It originated in the Bell-LaPadula model; different versions of it exist in several models.

Subject: A process or user; a participant in an access relation. A subject is a channel for information flow between the objects to which it has access.

Trusted: A component of a secure system is said to be trusted if some aspect of its processing must be relied upon by users or other components of the secure system to enforce the required policy. A user is often regarded as trusted for all purposes with objects which that user is authorized to read.

Trusted Process: A process, such that some aspect of its operation is critical to security. It may be relied upon to accomplish a task for or provide information to a security kernel (e.g., authentication) or it may be granted privileges to override an aspect of the access policy in a controlled way (e.g., for downgrading or multilevel object handling). It may be responsible for output marking; or it may alter kernel data.

User: A source or sink of information at the external interface of a computer system. Usually, but not always, a human being.